Patrones de diseño

Charla 2: Patrones para aplicaciones web. *Modelo-Vista-Controlador*

© 2006-2007 Depto. Ciencia de la Computación e IA

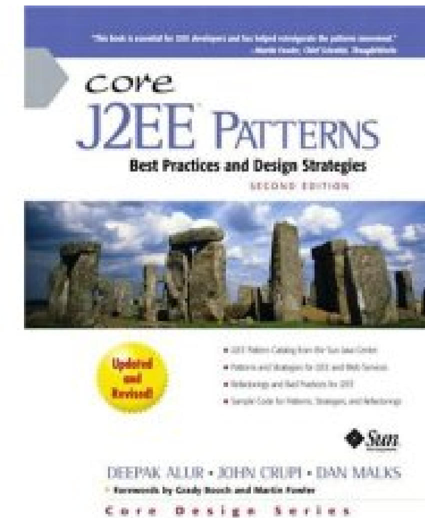
¿Qué veremos hoy?

- Patrones para aplicaciones web
  - no distribuidas (no EJBs)
  - servidores web Java (ej. Tomcat)
- Ya conocéis unos cuantos aplicables
  - Data Access Object
  - Transfer Object
  - Façade, Singleton, Factory,...
- Falta sobre todo la **capa de presentación**
  - **MVC: el más importante con diferencia**

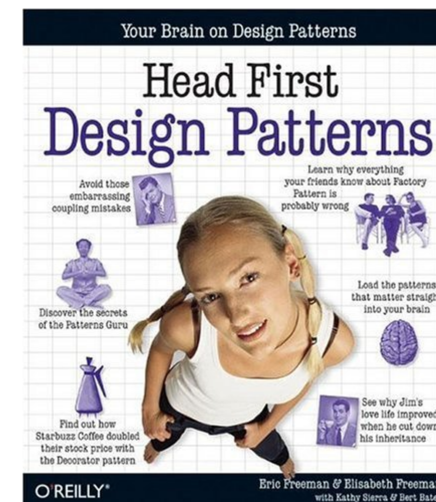
© 2006-2007 Depto. Ciencia de la Computación e IA

Bibliografía

- *Core J2EE Patterns, 2nd Ed.*
    - *www.corej2eepatterns.com*



- *Head First Design Patterns*
    - *Explicación de MVC para dummies*



© 2006-2007 Depto. Ciencia de la Computación e IA

¿Qué es un JSP?

- Código Java en páginas HTML

```
<html>
  <head>
      <title>Mi primera página JSP</title>
  </head>
  <body>
      <h1> Hola, <%= request.getParameter("nombre") %>
      Hoy es: <%= new java.util.Date() %> </h1>
  </body>
</html>
```


© 2006-2007 Depto. Ciencia de la Computación e IA

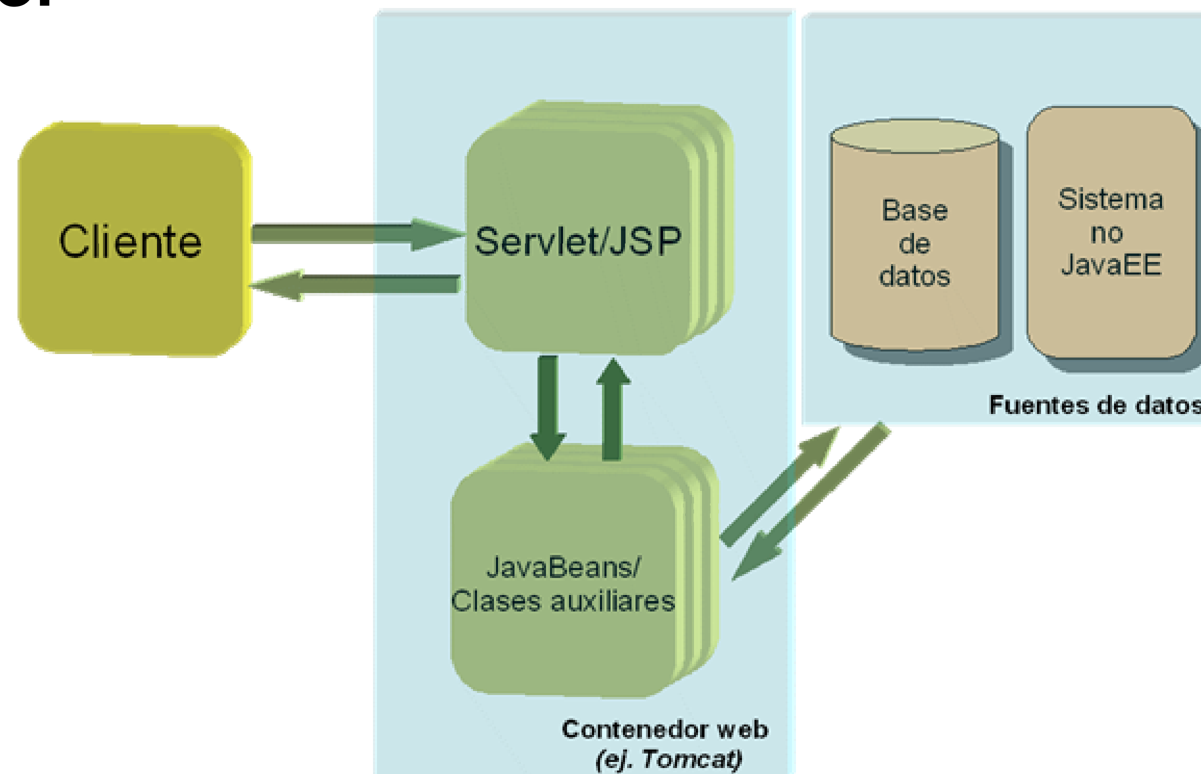
Arquitectura "naïf" para aplicaciones Web

- Todo el código va dentro de los servlets/JSP
  - Toma de parámetros HTTP
  - Implementación de las operaciones de "lógica de negocio"
  - Acceso a bases de datos
  - …
- Productivo para aplicaciones pequeñas, pero también inmanejable a medida que la aplicación crece

© 2006-2007 Depto. Ciencia de la Computación e IA

Modelo 1

- "Sacar" código fuera de los servlets/JSPs. En *Core J2EE Patterns* esto es el patrón **View Helper**



© 2006-2007 Depto. Ciencia de la Computación e IA

¿Qué es un JavaBean?

```
public class Usuario {
    private String nombre;
    private boolean varon;
    private int visitas;
    public UsuarioBean() { ... }
    public String getNombre() { return nombre; }
    public boolean isVaron() { return varon; }
    public int getVisitas() { return visitas; }
    public void setNombre(String n) { nombre=n; }
    public void setVaron(boolean b) { varon=b; }
    public void setVisitas(int v) { visitas=v; }
}
```

Campos privados (nombre, varon, visitas)

Constructor sin parámetros (UsuarioBean())

Métodos getXXX/isXXX sin parámetros (getNombre, isVaron, getVisitas)

Métodos setXXX (setNombre, setVaron, setVisitas)

© 2006-2007 Depto. Ciencia de la Computación e IA

¿Qué tal se llevan JSPs y JavaBeans?

```
<jsp:useBean id="usu" class="beans.Usuario" scope="session/>"
<html>
  <head>
     <title>¡Nos llevamos perfectamente!</title>
  </head>
  <body>
     <h1> Hola, ${usu.nombre}  </h1>
     <p> Ya has estado aquí ${usu.visitas} veces </p>
  </body>
</html>
```


© 2006-2007 Depto. Ciencia de la Computación e IA

Ejemplo de *View Helper con JavaBean*

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"  %>
<jsp:useBean id="miDAO" class="jtech.UsuariosDAO" scope="page"/>
<html>
  <head>Esto podría ser un listado de usuarios</head>
<body>
  <c:forEach items="${miDAO.listaUsuarios}" var="u">
     ${u.nombre} <br>
     ${u.visitas} <br>
  </c:forEach>
</body>
</html>
```


© 2006-2007 Depto. Ciencia de la Computación e IA

Combinación de servlets y JSP

Cliente
Servlets
JavaBeans
JSPs
Contenedor web
(ej. Tomcat)
Base de datos
Sistema no JavaEE
Fuentes de datos

1. Por cada caso de uso hay un servlet
2. El servlet llama a un JavaBean
3. El JavaBean ejecuta negocio/DAOs
4. El servlet le pasa el control a un JSP
5. El JSP muestra los resultados

© 2006-2007 Depto. Ciencia de la Computación e IA

Modelo-Vista-Controlador (MVC)

Cliente

<<Servlet>>
Controlador

Modelo

<<JSP>>
Vista

Contenedor web
(ej. Tomcat)

Base de datos

Sistema no JavaEE

Fuentes de datos

Cambia:
- La denominación
- Solo hay 1 servlet

© 2006-2007 Depto. Ciencia de la Computación e IA

Frameworks MVC

- Un framework proporciona una base sobre la que desarrollar nuestra aplicación
- En J2EE hay multitud de frameworks que implementan MVC:
  - Struts (el más conocido, un estándar de facto)
  - JavaServer Faces (centrado en la “vista”)
  - Spring
  - Webwork,…
- Es mejor usar un *framework* ya desarrollado que una implementación propia, aunque aquí veremos un ejemplo propio simplemente por motivos didácticos

© 2006-2007 Depto. Ciencia de la Computación e IA

Controlador

- Será un servlet
- En función de la petición, el controlador dispara la acción apropiada
  - En nuestro caso, llamando al método ejecutar de una clase que implemente la interfaz Accion
- Hay que
  1. Mapear las peticiones HTTP al servlet
  2. Indicar en el servlet qué acción hay que realizar para cada petición
  3. Definir las clases Java correspondientes a las acciones (≈1 clase por caso de uso)

© 2006-2007 Depto. Ciencia de la Computación e IA

Controlador: **1. Mapear las peticiones**

- Las peticiones *.mvc van a parar al servlet. Por ejemplo, login.mvc no será una página, sino que indicará que hay que disparar una acción

```
<servlet>
	<servlet-name>Controlador</servlet-name>
	<servlet-class>mvc.controlador.Controlador</servlet-class>
</servlet>
<servlet-mapping>
	<servlet-name>Controlador</servlet-name>
	<url-pattern>*.mvc</url-pattern>
</servlet-mapping>
```


© 2006-2007 Depto. Ciencia de la Computación e IA

Controlador: **2. Asociar petición con acción**

- En nuestro caso, se meten en el código. Normalmente estarán en un fichero de configuración

```
public void init () throws ServletException {
        acciones = new HashMap();
        acciones.put("prueba", new AccionPrueba());
        acciones.put("login", new AccionLogin());
        ...
}
```


© 2006-2007 Depto. Ciencia de la Computación e IA

Controlador: **3. Definir acciones**

- En nuestro caso, son clases que implementan la interfaz Accion

```
public interface Accion {
        public String ejecutar(ServletContext sc, HttpServletRequest request,
                    HttpServletResponse response) throws MVCException;
            ...
}
```


© 2006-2007 Depto. Ciencia de la Computación e IA

Controlador: el ciclo de proceso

1. A partir de la URL de la petición, se obtiene el nombre simbólico de la acción (login.mvc → login)

1. A partir del nombre simbólico y el HashMap del servlet se obtiene una instancia de la clase que implementa la acción (login → AccionLogin)

1. Se ejecuta la acción, llamando a su método ejecutar
   - Ejecutar recibe la petición y la respuesta HTTP (***)

2. La acción coloca el resultado en algún ámbito accesible a la vista (request,session,…) y devuelve un valor que indica la siguiente vista a mostrar (“personal.jsp”)

© 2006-2007 Depto. Ciencia de la Computación e IA

(***) Sofisticación: patrón Context Object

- **Recogido en *Core J2EE patterns***
- **Independizar la acción del HTTP**: se encapsula lo relevante de request y response en clases propias (context objects)
- **Ventaja:** poder reutilizar las acciones para otros clientes no HTTP (ej. *rich clients* con RMI)
- Esto se hace en algunos *frameworks: por ejemplo* Struts 2.0

© 2006-2007 Depto. Ciencia de la Computación e IA

Código de ejemplo

```
public void doPost (HttpServletRequest request, HttpServletResponse response)
                                          throws ServletException, IOException   {
    String nomAccion;    Accion ac;    String nomVista = null;

    try {
            //obtener de la URL el nombre de la acción
            nomAccion = getNomAccion(request);
            //a partir del nombre, obtener la clase asociada a la acción
            ac = getAccion(nomAccion);
            //ejecutar la accion
            nomVista = ac.ejecutar(getServletContext(), request, response);
            //mostrar la vista asociada a la accion
            mostrarVista(nomVista, request, response);    }
    catch(MVCException e) {
           request.setAttribute("exception", e);
           mostrarVista(VISTA_ERROR, request, response); }
  }
```


© 2006-2007 Depto. Ciencia de la Computación e IA

Sofisticación: patrón Application Controller

- Reducir a la mínima expresión el rol del controlador
  - Tomar la URL y llamar a otra clase que se encargue de todo. Recoger los resultados
- Application controller
  - Mapeo URL-acción
  - Ejecución de la acción
  - Obtención de la vista

© 2006-2007 Depto. Ciencia de la Computación e IA

El modelo

- La responsabilidad de los frameworks MVC suele terminar aquí
- Acciones: suelen estar “acopladas” a la capa web. Si introducimos código en ellas, no será reutilizable para otros clientes
- Beans y clases adicionales: en ellas se implementará la “lógica de negocio” y el acceso a la base de datos

© 2006-2007 Depto. Ciencia de la Computación e IA

La vista

- En nuestro caso son páginas JSP
- Las páginas se limitan a mostrar los datos que las acciones han dejado en JavaBeans

```
<h1>Mensaje</h1>
<b>Fecha/hora: </b> ${mensaje.fechaCadena} <br>
<b>De:</b> ${mensaje.remitente} <br>
<b>Para:</b> ${mensaje.destinatario} <br>
<b>Asunto:</b> ${mensaje.asunto} <br>
${mensaje.texto} <br>
```


© 2006-2007 Depto. Ciencia de la Computación e IA

Variantes de MVC

- Lo que hemos visto sería la versión del patrón que en *Core J2EE patterns* llaman "**service to worker**"
  - En otras fuentes se llama "push" porque a la vista se le envían los datos
  - Es la implementada por Struts
- Variante: "**dispatcher view**"
  - En otras fuentes se llama "pull" porque la vista dispara la lógica de negocio y por tanto "tira" de los datos
  - Es la implementada por JSF

 © 2006-2007 Depto. Ciencia de la Computación e IA

Push vs. Pull

Struts application

Browser

Struts servlet

Action subclass

Business logic

JSP page

JSF application

Browser

JSF framework

JSF view

JSF Component

Business logic

Event handler

© 2006-2007 Depto. Ciencia de la Computación e IA

Business Delegate

- Es un caso concreto del patrón *Façade* que ya vimos en la primera charla
  - Es el punto de entrada a la capa de negocio desde la de presentación
- Clase Java que contiene 1 método por cada caso de uso (en general)
- Nos permitirá distribuir la aplicación sin cambiar la capa de presentación (acciones)

© 2006-2007 Depto. Ciencia de la Computación e IA